Daniele Ricciardelli

ECE 4310.01

Gabriel Kuri

March 28, 2024

**Project 2**

A black screen with white text

Description automatically generated**Part 1**

The output of vmstat shows multiple fields to observe, which are (in order from left to right):

Procs:

* r: processes currently using the CPU
* b: processes waiting for I/O

Memory (in kilobytes):

* swpd: virtual memory used by swap space
* free: idle memory
* buff: memory used by the kernel for buffers
* cache: memory used by the kernel for page cahce

Swap (kb per second):

* si: memory swapped **in from** disk
* so: memory swapped **out to** disk

IO:

* bi: blocks **received** from a block device
* bo: blocks **sent** to a block device

System:

* in: interrupts per second
* cs: context switches per second

CPU:

* us: percentage of CPU time spent in running user processes
* sy: percentage of CPU time spent in running kernel processes
* id: percentage of CPU time spent idle
* wa: percentage of CPU time spent waiting for I/O operations to complete
* st: percentage of CPU time stolen by virtualization

When it comes to free, its output shows both the physical and virtual memory.

**Physical Memory:**

* 8,098,984 kB total memory
* 1,313,236 kB used by the system and applications
* 2,264,952 kB unused and free to use
* 419,104 kB shared among processes
* 4,520,796 kB allocated to the buffer and cache
* 6,061,560 kB available to be allocated to processes without having to swap data

**Virtual Memory:**

* 4,194,300 kB total memory
* 28,672kB used memory
* 4,165,628kB unused and free to use

**A black screen with white lines

Description automatically generatedA screenshot of a computer

Description automatically generatedPart 2:**

When ran on VS enterprise 2022 (personal approach)

When ran on Putty (assigned approach)

A screenshot of a computer

Description automatically generated

Notes:

*I used C++ instead of C#.* *In order to convert my code to C# I would only need to need to implement the following:*

A computer screen shot of white text

Description automatically generated

And delete the #define lines.

***The code can still compile on C anyways, just giving a heads-up of my approach.***

Moreover, I used scanf\_s() instead of scanf() as my compiler (*Visual Studio Enterprise 2022*) recommended it due to “safety” reasons or otherwise it wouldn’t compile. This modification protects/ensures the handling of character strings and buffer sizes when reading a string with %s.

This was “corrected” (altered) in the submission to the Unix server by changing the scanf\_s to scanf.

For convenience I shared screenshots of the code in VS and shared (copy-pasted) the code submitted to the Unix server.

The order of these two were also followed to create the project, thus the VS screenshots before sharing the code submitted through nano command.

I will attach to the email both codes in .txt format in case screenshots or paste wasn’t clear enough.

**Screenshots of the code in VS:**

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

**Copy of the code submitted to the Unix server using nano**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PAGES 100

#define PAGES\_DATA 4

void print\_Letters(int X) {

// Ensure X is within the limit [1, 10]

if (X < 1 || X > 10)

{

printf("X is out of bounds. It should be between 1 and 10.\n");

scanf(""); // Prevents instant closure of CMD - Note: This is not standard C and will likely do nothing. Consider using a character read instead.

return;

}

// Print the letter, starting from 'A' and add a space.

for (int i = 0; i < X; ++i)

{

printf(" %c ", 'A' + i);

}

printf("\n");

}

int main() {

// Size input in MB

int physical\_Memory\_Size, logic\_Memory\_Size, page\_Size = 0;

printf("Note: All size input by the user will be considered in MB\n\n");

printf("Enter physical memory size: ");

if (scanf("%d", &physical\_Memory\_Size) != 1 || physical\_Memory\_Size <= 0) // User validation

{

printf("\nInvalid input. Only positive integers will be considered.\n");

scanf("");

return 1; // Breaks

}

// Input for logical memory size

printf("Enter logical memory size: ");

if (scanf("%d", &logic\_Memory\_Size) != 1 || logic\_Memory\_Size <= 0)

{

printf("\nInvalid input. Only positive integers will be considered.\n");

scanf("");

return 1;

}

// Input for page size

printf("Enter page size: ");

if (scanf("%d", &page\_Size) != 1 || page\_Size <= 0)

{

printf("\nInvalid input. Only positive integers will be considered.\n");

scanf("");

return 1;

}

// Validate that the page size does not exceed the memory size

if (page\_Size > physical\_Memory\_Size || page\_Size > logic\_Memory\_Size)

{

printf("\nPage size cannot be larger than the physical or logical memory size.\n");

scanf("");

return 1;

}

// Math logic for total pages n frames

int total\_Pages = logic\_Memory\_Size / page\_Size;

int total\_frames = physical\_Memory\_Size / page\_Size;

int page\_Table\_Frames [MAX\_PAGES];

char logic\_memory\_data [MAX\_PAGES][PAGES\_DATA];

char frame\_table\_data [MAX\_PAGES][PAGES\_DATA];

// Tables initialization

for (int i = 0; i < MAX\_PAGES; i++)

{

page\_Table\_Frames[i] = -1;

for (int j = 0; j < PAGES\_DATA; j++)

{

logic\_memory\_data [i][j] = -1;

frame\_table\_data [i][j] = -1;

}

}

// Page to frame mapping input

int t\_Frame = 0;

char t\_Data [PAGES\_DATA] = { 0 };

int temp = 0;

for (int i = 0; i < total\_Pages; i++)

{

printf("Enter frame mapping for page %d: ", i);

scanf("%d", &t\_Frame);

page\_Table\_Frames[i] = t\_Frame;

// function implemented for fun as PAGES\_DATA value is defined and not an input

if (temp < 1)

{

printf("---------------------------------------\n");

printf("Example for formatting");

print\_Letters(PAGES\_DATA); // same as print\_Letters(4); (for this scenario)

printf("---------------------------------------\n");

temp++;

}

printf("Enter %d data value/s for page %d (separated by a space): ", PAGES\_DATA, i);

for (int j = 0; j < PAGES\_DATA; j++)

{

if (scanf(" %c", &t\_Data[j]) != 1) // User validation

{

printf("\nInvalid input. Only character data is permitted.\n");

return 1;

}

logic\_memory\_data[i][j] = t\_Data[j];

frame\_table\_data[t\_Frame][j] = t\_Data[j];

}

}

// Print logical memory

printf("\nLogical Memory:\n");

printf(" Page\t|Address| Data\n");

printf("-------------------------\n");

for (int i = 0; i < total\_Pages; i++)

{

for (int j = 0; j < PAGES\_DATA; j++)

{

if (logic\_memory\_data[i][j] != -1)

{

if (j == 0)

{

printf("%d\t| %d\t| %c\n", i, PAGES\_DATA \* i, logic\_memory\_data[i][j]);

}

else

{

printf("\t| \t| %c\n", logic\_memory\_data[i][j]);

}

}

else

{

if (j == 0)

{

printf("%d\t| %d\t| -\n", i, PAGES\_DATA \* i);

}

else

{

printf("\t| \t| -\n");

}

}

}

printf("-------------------------\n");

}

// Print page table

printf("\nPage Table:\n");

printf(" Page\t| Frame\n");

printf("----------------\n");

for (int i = 0; i < total\_Pages; i++)

{

printf(" %d\t| %d\n", i, page\_Table\_Frames[i]);

}

printf("----------------\n");

// Print frame table

printf("\nFrame Table/Physical Memory:\n");

printf(" Frame\t|Address| Data\n");

printf("-------------------------\n");

for (int i = 0; i < total\_frames; i++)

{

for (int j = 0; j < PAGES\_DATA; j++)

{

if (frame\_table\_data[i][j] != -1)

{

if (j == 0)

{

printf("%d\t| %d\t| %c\n", i, PAGES\_DATA \* i, frame\_table\_data[i][j]);

}

else

{

printf("\t| \t| %c\n", frame\_table\_data[i][j]);

}

}

else

{

if (j == 0)

{

printf("%d\t| %d\t| -\n", i, PAGES\_DATA \* i);

}

else

{

printf("\t| \t| -\n");

}

}

}

printf("-------------------------\n");

}

return 0;

}